1.**Write a Solidity contract that declares a variable, performs some basic arithmetic operations, and logs the result using an event.**

Solution:

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.18;

contract arithmeticOp{

function additionFunc(int \_data1, int \_data2) public pure returns(int){

return \_data1 + \_data2;

}

function subtrctionFunc(int \_data1, int \_data2) public pure returns(int){

return \_data1 - \_data2;

}

function multiplicationFunc(int \_data1, int \_data2) public pure returns(int){

return \_data1 \* \_data2;

}

function divisionFunc(int \_data1, int \_data2) public pure returns(int){

return \_data1 / \_data2;

}

function remainderFunc(int \_data1, int \_data2) public pure returns(int){

return \_data1 % \_data2;

}

function incrementerFunc(int \_data1) public pure returns(int){

return ++\_data1;

}

function decrementerFunc(int \_data1) public pure returns(int){

return --\_data1;

}

}

**2. a Solidity contract that includes a function with a conditional statement. The function should take an input parameter, perform some calculations based on the condition, and return the result.**

**Solution:**

// Minimum number of notes for given amount(notes containing - 1000, 500, 100, 50 ,20, 10, 1)

// SPDX-License-Identifier: GPL-3.0

pragma solidity ^0.8.18;

contract numberOfNotes{

function countingNotes(uint \_money) public pure returns(uint){

uint noNotes = 0;

if(\_money >= 1000){

noNotes += \_money / 1000;

\_money = \_money % 1000;

}

if(\_money >= 500){

noNotes += \_money / 500;

\_money = \_money % 500;

}

if(\_money >= 100){

noNotes += \_money / 100;

\_money = \_money % 100;

}

if(\_money >= 50){

noNotes += \_money / 50;

\_money = \_money % 50;

}

if(\_money >= 20){

noNotes += \_money / 20;

\_money = \_money % 20;

}

if(\_money >= 10){

noNotes += \_money / 10;

\_money = \_money % 10;

}

noNotes += \_money;

return noNotes;

}

}

**3.Write a Solidity contract that uses a mapping to store and retrieve data. Include functions to add, update, and retrieve data from the mapping.**

Solution:

// SPDX-License-Identifier: SPL-3.0

pragma solidity ^0.8.18;

contract mappingContract{

mapping (uint => string) public register;

mapping (uint => bool) public verify;

function add(uint \_rollNo, string memory \_name) public {

register[\_rollNo] = \_name;

verify[\_rollNo] = true;

}

function update(uint \_rollNo, string memory \_name) public {

register[\_rollNo] = \_name;

}

function retrive(uint \_rollNo) public view returns(string memory){

if(verify[\_rollNo])

return register[\_rollNo];

return "not exists";

}

}

**4.Write a Solidity contract that uses an array to store a list of addresses. Include functions to add, remove, and retrieve addresses from the array.**

**Solution:**

// SPDX-License-Identifier: SDL-3.0

pragma solidity ^0.8.18;

contract arrayWithAddress{

address[] addressArray;

function addEle(address \_add) public {

addressArray.push(\_add);

}

function retriveEle(uint \_index) public view returns(address){

return addressArray[\_index];

}

function delEle(address \_add) public {

for(uint i = 0;i < addressArray.length - 1;i++){

if(\_add == addressArray[i]){

addressArray[i] = addressArray[addressArray.length - 1];

}

}

addressArray.pop();

}

}